CSE Senior Design

Detail Design Specification

Department of Computer Science and Engineering
The University of Texas at Arlington

Team: BetaHomes

Project: Z-Wave Smart Home

Team Members:
Nicholas Earwood
Benjamin Frank
Andrei Patapau
Aerina Shrestha
Santosh Upadhyay

Last Updated: 7/27/12 11:15

Table of Contents

Document Revision History

Revision Number	Revision Date	Description	Rationale
0.1	6/29/12	First Rough Draft	A rough draft is required.
0.2	7/3/12	Added several sections	Added sections required for completeness.
0.3	7/9/12	Added more sections	Added sections required for completeness.
0.4	7/10/12	Corrections made by peer review	Peer review improved the document.
0.5	7/11/12	DDS gate review	Complete document for gate review
0.6	7/26/12	DDS gate review	Peer review improved the document.
0.7	7/27/12	Consistencies, spelling, grammar	Peer review comments.
1.0	7/27/12	Baseline	All requested corrections made.

List of Figures

List of Tables

1. Z-Wave Smart Home Introduction

1.1 Document Overview

The Detailed Design Specifications will provide further detail based on the Architectural Design Document. The ZSH (Z-Wave Smart Home) system was earlier divided into layers that contained various subsystems. These subsystems are now decomposed into modules that provide minute details including the pseudo code. The document also provides the relationship between the various modules and the requirements traceability. Finally, Quality Assurance will cover the testing considerations and Acceptance Plan will ensure that the product is consistent with the consumer's expectations.

1.2 Product overview

This section provides an overview of the Z-Wave Smart Home (ZSH) product. The ZSH product is designed to operate Z-Wave enabled devices by allowing the user to control these devices remotely or via schedules/rules. ZSH is not a decision making product, the system cannot decide what it will and won't do. Control of the ZSH system is always in the hands of the user and the system will only operate within the parameters defined by the user. The ZSH design is a step toward the future of home independent automation but is still far from an Artificially Intelligent home. For product visual concept see figure 2-1.

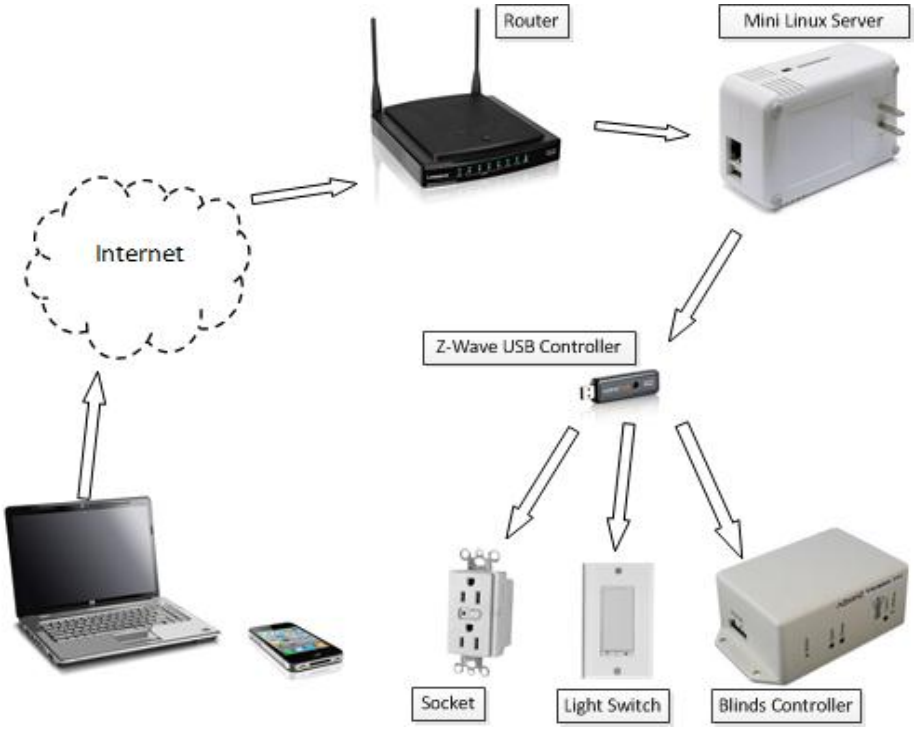


Figure 1-1: Product visual concept

1.2.1 ZSH Features/Components:

Linux Server – The main component of the ZSH product. The server will communicate with Internet browsers as well as store schedules/rules as defined by the user. The server will command the Z-Wave adapter. The server will host the web-interface.

Z-Wave adapter – Translates communications with all Z-Wave enabled devices and the server within a minimum 17 meter radius. The adapter is either embedded or plugs into the server (typically via USB). The adapter is controlled by the server.

Internet – The Internet will be required at all times to operate the ZSH product remotely. It is required for communication between the smart phone controller and ZSH server.

Web Interface – The server will host the web-interface. The web-interface will be the main form of user interaction with the ZSH product. The web-interface will be database driven, allowing the user to store schedules and rules for the system to execute.

Router/Modem – Communication transition between the Internet and the ZSH server. This is needed by the ZSH, but is not part of our product.

Z-Wave modules – Controlled through the server/adapter. The modules in this ZSH product will consist of light switches, electrical sockets, and a Blinds Control Unit. These modules will communicate with each other and the adapter.

Blinds Control Unit (BCU) – Communicates with the adapter and other Z-Wave modules. Used to open and close household blinds via changes in blind's slat angles. BCU uses rechargeable batteries to operate. BCU uses solar panels to recharge batteries. BCU is wall mounted and designed to retrofit many blind types.

Database – The database is maintained on the server. The database stores rules and schedules created via user interaction with the web-interface.

User Interface (UI) – The UI will consist of computers and mobile devices. The UI will require Internet access and the ability to communicate via HTTP. The UI will interact with the Web-Interface. This will be the main form of user control over the ZSH system.

The ZSH product will consist of the server, adapter, Z-Wave modules, and Blinds Control Unit (BCU). The server is a small, self-contained PC that plugs directly into a wall outlet, preferably close to Z-Wave enabled devices and the router. The adapter is typically a USB Z-Wave adaptor, but some are embedded in the PC itself. Either way, for our purposes the server and adapter can be thought of as a single component. The Z-Wave modules can be any number of Z-Wave enabled devices but the ZSH product will provide only a light switch, electrical socket, and BCU. The light switch and electrical socket would replace a standard light switch or electrical socket and should be installed according to the directions

found with the purchase of those items. The BCU is a device that is no larger than 30cm x 30cm x 15cm and weigh no more than 1kg. The BCU is a wall mounted unit that the user would need to screw into the wall near the rod that controls the angle of the blind's slats. The rest of the components listed above are separate from the ZSH product but are still required for its use.

1.2.2 Product purpose

The purpose of the Z-wave Smart Home (ZSH) product is to control electronic devices in the home remotely and wirelessly. Using the ZSH product should make controlling electronic devices in the home more convenient for the user. The ZSH product is not a replacement for appliance controllers such as Television and DVD players but instead is a method for controlling the power on and off of such devices. Typical use of the ZSH product would be using a smart phone with Internet connection to logon to a web server. The web server then controls the Z-wave enabled devices in the House. The user is expected to use this method of access to directly control appliances or create schedules/rules for automated control.

1.2.3 Scope

The Z-Wave Smart Home (ZSH) product is designed to be a practical method of control for people who would enjoy greater control and ease of use of household appliances. This includes persons with appliances in normally hard to reach places. The ZSH product would also accommodate persons with disabilities as they can control frequently used appliances without being physically near said appliances. These appliances could include but are not limited to coffee makers and toasters. The ZSH device is web enabled for easy access through a smart phone or personal computer both of which the intended audience are required to own for access to all the features of the ZSH product.

1.2.4 Definitions and Terms

SH: Smart home
SHS: Smart home system
DDS: Detailed design specifications
ZSH: Z-Wave Smart Home
ADS: Architectural Design Specifications
DDS: Detailed Design Specifications
SRS: System Requirements Specification
API: Application Programming Interface
UI: User Interface
GUI: Graphical User Interface
BCU: Blinds Control Unit
OS: Operating System

2. Architecture Overview

2.1 Overview

The system consists of three layers: System Control Layer, User Layer, and Device Layer. The System Control Layer controls the major functionalities of the system. It is responsible for starting up the system, connecting the system with the Z-Wave devices, managing the database, and establishing and maintaining the communication of the system via internet/intranet. On the other hand, User Layer creates the User interface for creating, setting, and controlling the Z-Wave devices within the system. Device Layer is responsible for hardware setup of the Z-Wave devices such that they can interact with System Control Layer and User Layer to control the Z-Wave devices.

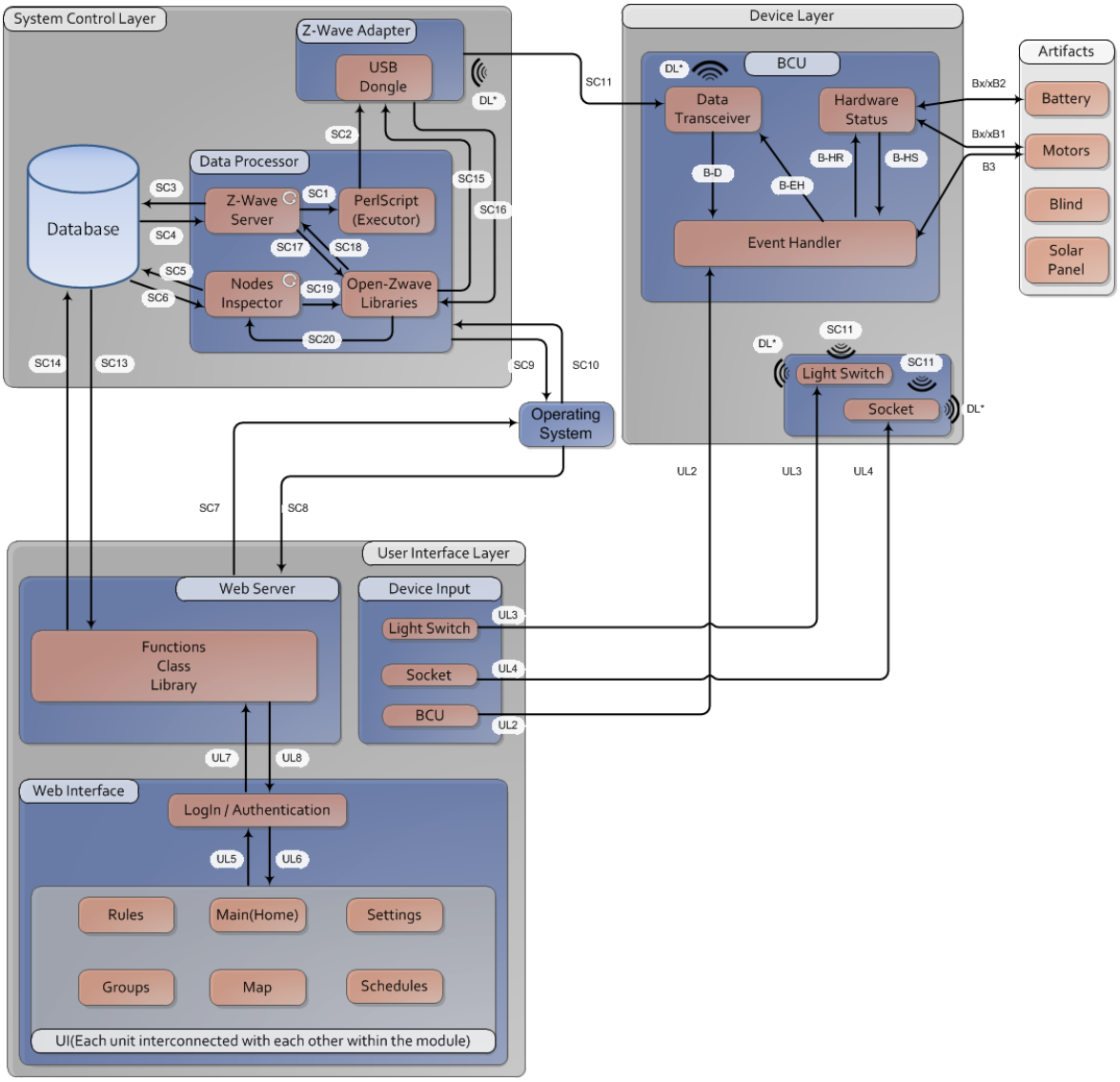


Figure 2-1: Architecture Overview

Data E21lement	*Description of Data Element*
SC11	Broadcasted receiving RF frames including packaged commands
DL*	Broadcasted transmitted RF signals consisting all the network ids
B-D	Commands to raise/lower blinds, or tilt the slat, or check the status of battery.
B-HS	Recent status on battery level, and motor count.
B-HR	Request command on status of battery, and motors.
B-EH	Processed data on status of blind and battery.
Bx/xB1	To Hardware Status: Motors' count (integer). To Motor: Request to get the motor count.
Bx/xB2	To Hardware Status: Battery voltage (float). To Motor: Request to get the battery voltage.
UL2	Manual command to control the blinds
B3	Request to increment/decrement motors' count.
UL3	Command to control the light switch
UL4	Command to control the socket
SC1	String type executive command
SC2	Serialized commands that needs to be carried to the Z-wave devices(action)
SC3	Formatted queries for appropriate commands
SC4	Formatted data results after running queries
SC5	Formatted queries for appropriate commands
SC6	Formatted data results after running queries
SC7	Resource requests from Operating System
SC8	Resource allocation
SC9	Resource requests from Operating System
SC10	Resource allocation
SC11	Command signals to control Z-Wave modules(Light, Socket or BCU)
SC12	Transaction response to communicate with the web interface
SC13	Formatted data results after running queries
SC14	Formatted queries for appropriate commands
SC15	Serialized commands that needs to be carried to the Z-wave devices(status)
SC16	Serialized data (status and acknowledgements) from Z-wave controller for further processing
SC17	Formatted node status request
SC18	Formatted node status response
SC19	Formatted node status request
SC20	Formatted node status response
UL5	Request a function call
UL6	Output formatted data to user interface
UL7	Process authenticated function call
UL8	Formatted data result after running a function call

Table 2-1: Data Flows

2.2 Layer Descriptions

2.2.1 User Interface Layer

The User Interface Layer shall be designed to translate interactions between a user and the system. It shall contain two major interface categories: software (logical) – GUI (web interface) and hardware (physical) – manual device interactions. Example: Turn the lights on/off using switch button. Web interface shall provide compatibility with common PC browsers supporting HTML/XHTML format and mobile devices, specifically the iPhone (Safari).

The User Interface Layer shall be designed to handle all interactions between a user and the system, and display session data (applies to web interface only). The main goal is an effective operation and control of the system, and feedback from the system which aids the operator in making operational decisions. It shall provide a means of:

Input – allow system manipulation

Output – allow the system to indicate the effects of the user's manipulation.

2.2.2 System Control Layer

The system control layer is the core of the SHS. It is responsible for the web server, database management, and command translation between the Z-Wave adapter and the web server (and thus, the user interface).

The System Control Layer (SCL) interfaces with the user interface via the web server subsystem. This data is handled to and from the data processor, which is a large subsystem that handles most of the logic of the SH controller. Scheduling, rules, device controls and feedback are all handled by the data processor. The data processor uses data from the Z-Wave subsystem and database subsystem to make decisions and output to both the web server (user interface) and Z-Wave output. The Z-Wave subsystem is the sole channel to communicate with Z-Wave devices. It includes the hardware Z-Wave adapter. The Z-Wave, web server, database and data processor are the four distinct and critical parts of the SCL.

2.2.3 Device Layer

The Device Layer contains the Blinds Control Unit (BCU), Z-Wave light switch, and Z-Wave electrical socket. These devices makeup a large portion of the hardware components found in the Z-Wave Smart Home (ZSH) System. This layer handles input from the User Layer and status communication to the System Control Layer. The purpose of the Device Layer is to control Z-Wave devices via manual interface or Z-Wave communication and perform an operation based on received commands.

The purpose of the Device Layer is to control Z-Wave devices via manual interface or Z-Wave communication and perform an operation based on received commands.

3. System Control Layer

The system control layer is the core of the Smart Home System. It is responsible for the web server, database management, and command translation between the Z-Wave adapter and the web server (and thus, the user interface).

It consists of three subsystems: Database, Z-Wave Adapter, and Data Processor. The Data Processor subsystem includes four modules: Z-Wave Server, Nodes Inspector, PerlScript and Open-Z-Wave Libraries.

Scheduling, rules, device controls and feedback are all handled by the Data Processor. The data processor uses the data from Z-Wave Adapter subsystem and Database subsystem to make decisions and output to both the web server (through Database) and Z-Wave Adapter, which in turn passes the serialized signal to Z-Wave modules. The Z-Wave Adapter subsystem is the sole channel to communicate with Z-Wave devices.

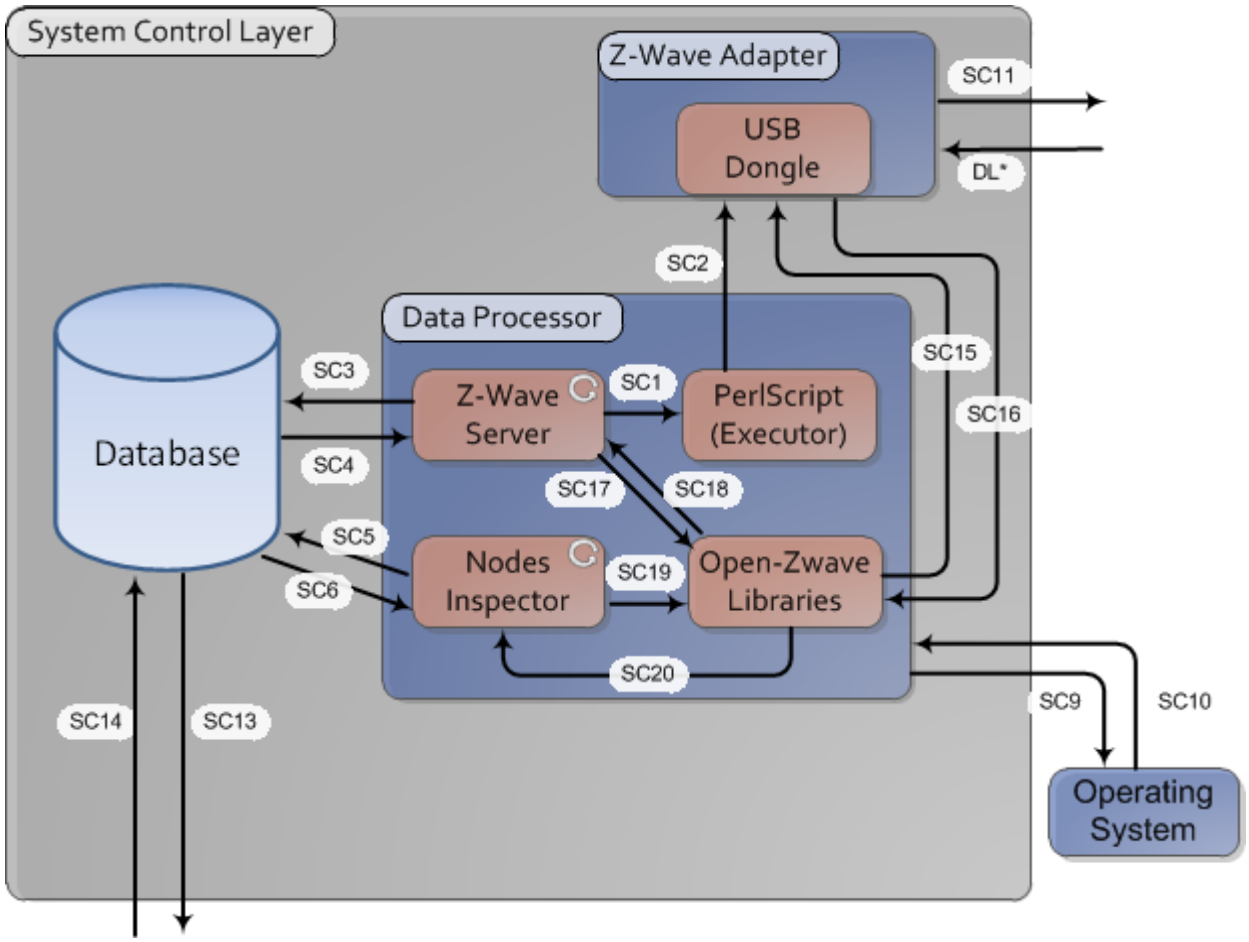


Figure 3-1: System Control Layer

3.1 Z-Wave Adapter Subsystem

3.1.1 USB Dongle Module

The version of adapter used for the project is the Aeon Labs Z-Stick Series 2. Is a self-powered Z-Wave USB dongle with push button for remote network creation (independent from external power and host microprocessor).When attached to a host processor, it becomes a Z-Wave communication device, which exposes the Zensys API (SerialAPI) through integrated USB. This device is meant primarily to allow a host processor to control up to 232 Z-Wave devices using the Z-Wave technology protocol.

The Aeon Labs Z-Stick is easily upgradeable by the end-user such that the latest ZWave protocols and commands are always available. It can be upgraded in field via its USB port which also serves as a charging port for its internal battery.

3.1.1.1 Interfaces

The hardware interface specifications are proprietary and are therefore inherently encapsulated.

3.1.1.2 Physical data structure/data file descriptions

The module is dependent on the Open-ZWave Libraries which are reliant on Linux OS services and Device Layer radio frequency signals.

Internal data descriptors:

- Radio transceiver/receiver
- Microprocessor
- 32kB flash memory, containing the Z-Wave protocol and the application
- System interfaces, including digital and analogue interfaces to connect external devices such as sensors
- A 3DES engine to ensure confidentiality and authentication (100 series)
- Triac controller, to reduce the module cost of dimming applications

3.1.1.3 Process

N/A (The hardware interface specifications are proprietary and are therefore inherently encapsulated).

3.2 Database Subsystem

3.2.1 Database Module

The Database module is responsible for the interfacing that occurs between Data Processor subsystem and the Web Server. Its main functions are 1) to store data processed by Data Processor, which in turn

allows Web Server to pull this data, process it and pass it on to the User Interface Layer for display.
2) It accepts requests from Web Server which are being handled by Data Processor.

3.2.1.1 Interfaces

Interfaces with Web Server, Z-Wave Server and Nodes Inspector modules.

3.2.1.2 Physical data structure/data file descriptions

The following data tables will be used by Data Processor to store information regarding node/home network status, groups and command classes associated with each node.
To allow Foreign Key relation the database type will be InnoDB.
Character Set: utf8_unicode_ci – more accurate and supports various languages of non-Unicode format.

- command_classes
  - node_id
  - id
  - name
- home
  - id
  - home_id
  - md5_hash_of_xml
  - server_1
  - server_1_exit
  - reload_nodes
  - inspector
  - xml_remove
  - nodes_md5_db

- nodes_in_group
  - group_id
  - node_id
- nodes_c (current state)
  - node_id
  - manufacturer_id
  - product_type_id
  - product_id
  - 15oolean15urer_name
  - product_name
  - isON
  - level
- groups
  - group_id
  - name

- nodes_d (desirable state)
  - node_id
  - manufacturer_id
  - product_type_id
  - product_id
  - 15oolean15urer_name
  - product_name
  - isON
  - level

nodes_d table will be used for device manipulation. It will be handled by Web Server to write a node execution request and processed by Data Processor for action execution.

3.2.1.3 Process

Update table:

```
UPDATE `home` SET `home_id`='$getHomeId',`md5_hash_of_xml`='$getHash' WHERE id = 1
```

Insert data into table:

INSERT INTO `nodes`(`node_id`, `isON`, `level`) VALUES ('$nodeArray[$i1]', '$nodeArray[$i2]', '$nodeArray[$i3]')

Remove all data from table:

DELETE FROM `nodes` WHERE 1

Get node information:

SELECT * FROM _nodes_ WHERE node_id = '$curr_db_id'

```
--
-- Table structure for table `nodes`
--

CREATE TABLE IF NOT EXISTS `nodes` (
  `node_id` int(11) NOT NULL,
  `manufacturer_id` varchar(20) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL
DEFAULT 'NOT FOUND',
  `product_type_id` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL
DEFAULT 'NOT FOUND',
  `product_id` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL
DEFAULT 'NOT FOUND',
  `16oolean16urer_name` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT
NULL DEFAULT 'NOT FOUND',
  `product_name` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL
DEFAULT 'NOT FOUND',
  `isON` varchar(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT
'NOT FOUND',
  `level` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`node_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

3.3 Data Processor Subsystem

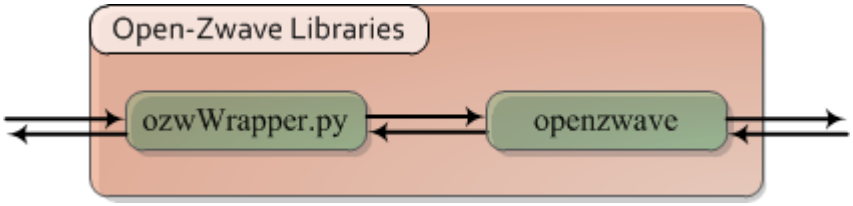
3.3.1 Open-Z-Wave Libraries

Open-Z-Wave is a library which provides an API for applications to access and control a Z-Wave controller connected to the PC via a serial or HID connection. For logical purposes ozwWrapper.py file is included in this module as well (see data description below).

3.3.1.1 Interfaces

Interfaces with USB Dongle, Z-Wave Server and Nodes Inspector modules.

3.3.1.2 Physical data structure/data file descriptions



Open-ZWave library structure:

config	XML files that provide device- and manufacturer-specific information, as well as the Z-Wave command class codes.
Cpp	The Open-ZWave library project.
Cpp/build	Makefiles and Visual Studio project/solution files for building the library under linux, Mac and Windows (VS2008 and VS2010).
Cpp/examples	A minimal console application (MinOZW) that can be built under linux, Mac and Windows. It is a useful example of how to start up the library and can be used to generate a log file (OZWlog.txt) to diagnose startup or other problems with the library and/or Z-Wave devices.
Cpp/hidapi	Files related to implementation of a "Human Interface Device" (HID) connection between the PC and the Z-Wave controller. While many controllers use a serial interface, others (the ThinkStick, for example) use HID via USB.
Cpp/lib	Essentially empty.
Cpp/src	The source code for the Open-ZWave Library. The top-level code is in this directory; subdirectories contain files to implement the command classes, the value classes and platform-specific code (linux, Mac and Windows).
Cpp/tinyxml	The TinyXML class code. TinyXML is used to read and write XML files for persistent storage.
Documents	Draft documentation for the library and example applications.
Dotnet	A .NET wrapper for the Open-ZWave library.
Dotnet/build	Project/Solution files for building Open-ZWaveDotNet under VS2008 and VS2010.
Dotnet/examples	Code for OZWForm, a .NET application that demonstrates use of Open-ZWaveDotNet to connect to a controller, read information about nodes, process notifications, etc.

dotnet/src	The Open-ZWaveDotNet wrapper code.
License	License documents.

ozwWrapper:

ozwWrapper is a python script which serves as an intermediary link between Open-ZWave libraries and applications that are rely on a data processed by Open-ZWave libraries and pulled out by ozwWrapper. Z-Wave Server and Nodes Inspector modules will communicate to Open-ZWave libraries directly through ozwWrapper.

3.3.1.3 Process

The Open-ZWave library creates an Open-ZWave::Driver, waits for connection and gets node information using provided by functions for data polling.

3.3.2 PerlScript (Executor)

Used to send out commands via USB adapter and allows adding devices to the Z-Wave network using the "zwave add" command.

3.3.2.1 Interfaces

Interfaces with USB Dongle and Z-Wave Server modules using Comprehensive Perl Archive Network(CPAN) – Socket Data Serialization. All of the executive commands will be processed and passed on to USB Dongle Module using PerlScript.

3.3.2.2 Physical data structure/data file descriptions

The module is dependent on Comprehensive Perl Archive Network.

Internal Data Description / Functions and their responsibilities:

receive() – read serial port data stream packets, send each to receive_once() for processing
receive_once() – read pending bytes from the serial port, ack if they look like a packet. Return true if we got an ack, false otherwise.
Transmit($data) – transmit one packet
mkreqpacket() – create request packet
packpack() – convert created packet into a string using zwave rules template
dim() – dimmer command function
switch() – switch command function(on/off)
addNode() – add a new node function
addNodeStop() – stop add node mode/function

handle_packet() – used for adding a node function. Forms add node request, halts and listens for a new node.

3.3.2.3 Process

```
# Shifts the first value of the array off and returns it
my $port = shift; //gets port number from cmd arguments

# Serial port initialization:
use Device::SerialPort;
my $serial_port = Device::SerialPort->new ($port,1);

# Based on cmd type implement if/elseif statement to identify command type call
```

receive() – read serial port data stream packets, send each to receive_once() for processing

```
sub receive {
    my( $timeout ) = @_;

    my $end = time+$timeout;
    $stopreceive = 0;
    do {
      receive_once();
    } while( ($end > time) && ! $stopreceive );
}
```

receive_once() – read pending bytes from the serial port, ack if they look like a packet. Return true if we got an ack, false otherwise.

Transmit($data) – transmit one packet

mkreqpacket() – create request packet

packpack() – convert created packet into a string using zwave rules template

```
sub packpack {
    my( @bytes ) = @_;

    my $seq = "";
    foreach my $byte (@bytes) {
      $seq .= pack( "C", $byte );
    }
    return $seq;
}
```

dim() – dimmer command function

```
sub dim {
    my( $unit, $level ) = @_;
    return( packpack( mkreqpacket( 0, 0x13, $unit, 3, 0x20, 1, $level, 5 ) ) );
}
```

switch() – switch command function(on/off)

```
sub switch {
    my( $unit, $onoff ) = @_;
    dim( $unit, $onoff ? 255 : 0 );
}
```

addNode() – add a new node function

```
sub addNode {
    return( packpack( mkreqpacket( 0, 0x4a, 0x01 ) ) );
}
```

addNodeStop() – stop add node mode/function

```
sub addNodeStop {
    return( packpack( mkreqpacket( 0, 0x4a, 0x05 ) ) );
}
```

handle_packet() – used for adding a node function. Forms add node request, halts and listens for a new node.

3.3.3 Nodes Inspector

(Contains: inspector.py, inspector.php and ozwWrapper.py)
Main purpose of this module is to scan the home nodes network every 3 hours and to check for the status of the nodes to make sure that all are alive and function properly. In case of one of the nodes ‘death’ it shall remove current node home network xml map. If that happens (xml file gets removed) Z-Wave Server will automatically rescan home network and recreate a new xml map with updated nodes information. This is not the same as getting node status!!!

3.3.3.1 Interfaces

Nodes Inspector module interfaces with Open-ZWave Libraries and Database.

Z-Wave Server vs Nodes Inspector

As mentioned above, 'Nodes Inspector' runs only once every 3 hours. It is not a server, but a separate, independent application which checks whether a node failure occurred. Z-Wave Server is not capable of determining whether a node has died or if it is still alive. The most recent node data stored in USB dongle flash memory, and if malfunction occurred and USB didn't receive status or any type of feedback from failed node, then it simply keeps old node status information without letting us know that feedback wasn't received. Of course, if no feedback received, then the timeout occurs.

The reason Nodes Inspector is a separate application is because even though we need to stop Z-Wave Server (just a python execution part), we still need to run the part of the server which takes care of the command execution. In other words – while Nodes Inspector runs, we won't be able to pull the most recent status of devices (for a period of approximately 50 seconds), but we still can execute any commands and run rules and schedules without interruption of service.

3.3.3.2 Physical data structure/data file descriptions

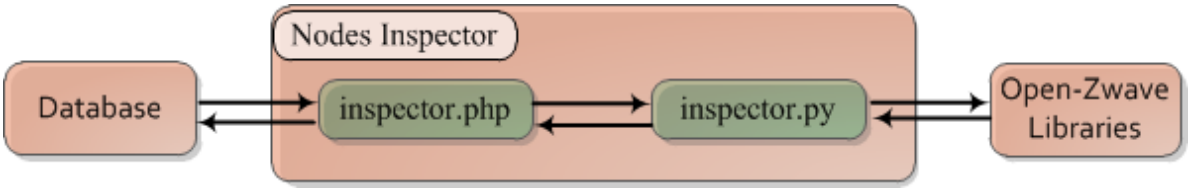


Figure 3.2: Nodes Inspector Module

inspector.php

db_connect() – establish connection with database

getPortNumber("fcall") – get a full path to USB adapter(also checks if adapter is exist)

comparator() – compares current node map with newly created

main() – main function that runs all the time in endless loop(brake option to exit out and stop is available)

inspector.py

Function: prfeed(device_path) – one and only function is to inherit some of the ozwWrapper.py functionality and initialize nodes on the network.

OS resources used:
time – provides various time-related functions
Datetime – supplies classes for manipulating dates and times
Sys – System-specific parameters and functions
os – Miscellaneous operating system interfaces
glob – Unix style pathname pattern expansion

3.3.3.3 Process

inspector.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# resource allocation and import of all neede libraries
import time
import datetime
import sys
import os
import glob
from common.ozwWrapper import ZwaveWrapper, ZwaveNode, ZwaveValueNode
from decimal import *
# ------------------------------------------------------------------------------+
# get USB adapter full path(path found and passed by inspector.php as a command
line argument)
device_path = sys.argv[1]


def prfeed(device_path):
      getcontext().prec = 3
      executionTime = Decimal(0)
      maxExecutionTime = Decimal(60) # Maximum initialization time (in seconds)
      sleepTime = Decimal(0.1)
      # ------------------------------------------------------------------------+
      # REMOVE zwcfg_*.xml FILE - DONE IN PHP BEFORE CALLING THIS FILE          |
      # ------------------------------------------------------------------------+
      w = ZwaveWrapper.getInstance(device = device_path, config = '../Open-
ZWave/config/')

      feedback = "INITIALIZATION SUCCESS"

      while bool(w.initialized) != True:
            executionTime = executionTime + Decimal(sleepTime)
            time.sleep(sleepTime)
            if executionTime > maxExecutionTime:
                  seq = ['INITIALIZATION FAILED | Execution time: ', executionTime,
' seconds']
                  feedback = ''.join(map(str, seq))
                  break

      return feedback

# return status of request - (Failed or Success)
return prfeed(device_path)
```

3.3.4 Z-Wave Server

(Contains: server_1.py, server_1.php and ozwWrapper.py)
Main purpose of this module is to scan the Z-Wave network, check for the status of nodes, update database, pass execution command to the perl script.

3.3.4.1 Interfaces

Z-Wave Server module interfaces with Open-ZWave Libraries, Database and PerlScript.

3.3.4.2 Physical data structure/data file descriptions

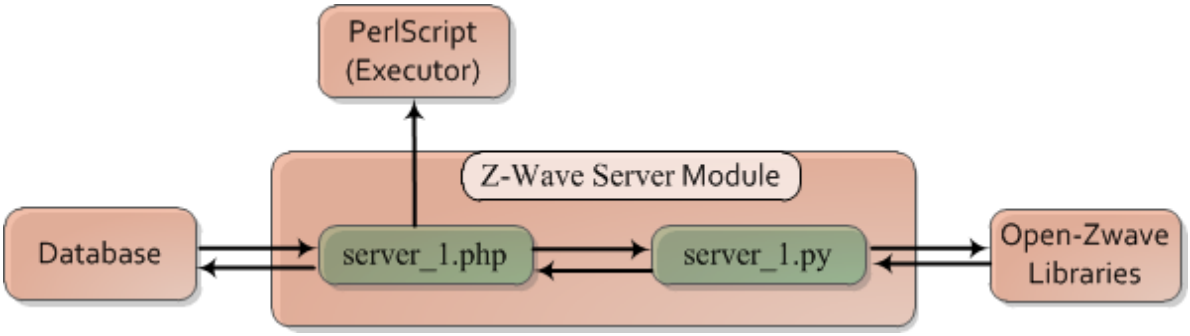


Figure 3-3: Z-Wave Server Module

server_1.php

main() – main function that runs all the time in endless loop(brake option to exit out or pause is available)

startLog() – creates and initializes log data file

db_connect() – establish connection with database

$port = getPortNumber("fcall") – get a full path to USB adapter(also checks if adapter is exist)

removeXML() – removes xml file in case of xml_remove request from database

reload_nodes() – remove all nodes from database and create new data table

exec_py($port) – execute server_1.py file. Pass full path of USB adapter found in getPortNumber function.

Load_db_data() – populate nodes table with data

getXMLData($node_id) – get nodes manufacture data from xml file per individual node

getXML_Home_id() – gets home id from xml file

getManufacturerSpecificXML() – gets manufacturer data for all nodes during creation

exec_command() – checks desirable state nodes table and compares with a current state. If difference found => execute action based.

Server_1.py

Functions:

initializeNodes() – initializes nodes on the network inheriting ozwWrapper.py functionality if xml file with nodes information is not created yet.

pullData() – pulls formatted node information data from the network and passes it to the server_1.php. One pull cycle takes about 2-3 seconds.

OS resources used:

time – provides various time-related functions
Datetime – supplies classes for manipulating dates and times
Sys – System-specific parameters and functions
os – Miscellaneous operating system interfaces
glob – Unix style pathname pattern expansion

3.3.4.3 Process

IF DB Current State != Current State (State Of Device)

    Update DB Current State & Desirable State

ELSE

    Do Nothing (Should be easy to implement)

IF DB Desirable State != DB Current State

    Execute Desirable State (through Perl)

ELSE

    Do Nothing

IF (SCHEDULE | RULE)

    Set DB Desirable State

ELSE

    Continue

4. User Interface Layer

The User Interface Layer is responsible for translation of interactions between a user and the system. It contains three subsystems: Web Server, Web Interface (GUI) and Device Input (Hardware Interface).

The main goal is an effective operation and control of the system, and feedback from the system which aids the operator in making operational decisions. It shall provide a means of:

- Input – allow system manipulation
- Output – allow the system to indicate the effects of the user's manipulation.

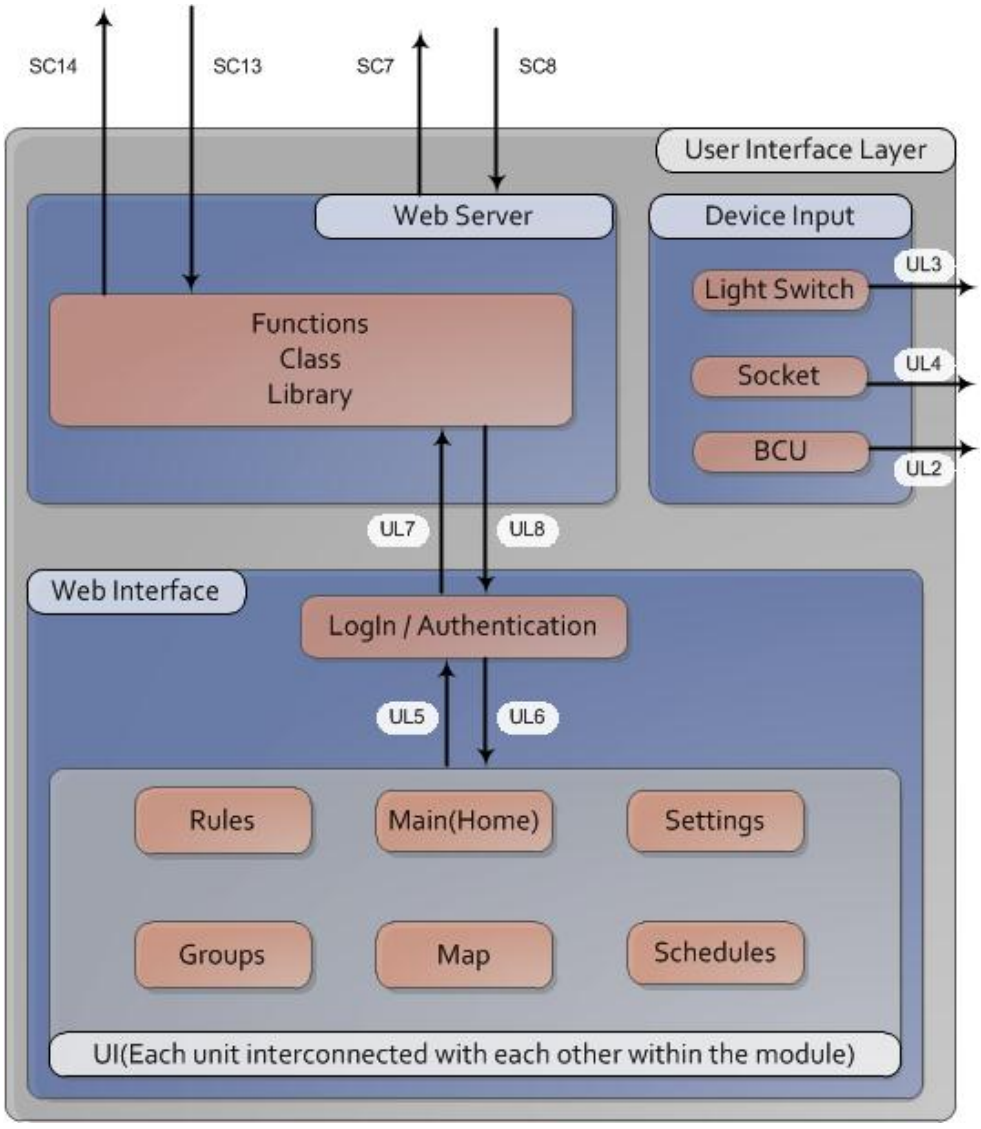


Figure 4-1: User Interface Layer

4.1 Web Server Subsystem

4.1.1 Functions Class Library Module

Function Class Library is a set of functions that are will be accessed from UI module in Web Interface Subsystem.

4.1.1.1 Interfaces

The Function Class Library interfaces with a database and UI through Login/Authentication checkpoint which checks for the status of user session (if user logged in). The web server itself will be reached by either static IP(default option), via free version of dynDNS(domain name is free but inconvenient to use – too long) or via professional version of dynDNS with a domain name of user's choice.

4.1.1.2 Physical data structure/data file descriptions

The module is dependent only on PHP native environment(PHP version is 5.3)

Internal Data Description / Functions and their responsibilities:

Login($u_name, md5($password)) – generates a new session if login and password are matched with the parameters in database(default username – admin, default password – admin)

List_nodes() – generates frame for each node with appropriate information for the node type(if switch type => display only On and Off) and displays it on Main page.

Voice() – provide voice API button t execute a voice command. Using Online Javascript speech recognition API Javascript and Flash API. Online Javascript speech recognition API Javascript and Flash API.

createMap() – medium priority requirement. Potential API to use is Scalable Vector Graphics. It is a language for describing 2D-graphics and graphical applications in XML and the XML is then rendered by an SVG viewer.

Add_rule() – creates a rule and adds it to database(Straight forward: ask to specify node id, action and node id and action of the node which will trigger the event)

Remove_rule() – removes created rule from database using a simple query call(DELETE FROM `rules` WHERE id =n)

List_rules() – loads all of the rules from database(just a 'select * from rules' query)

List_schedules() loads all of the schedules from database(just a simple 'select from schedules *' query)

Add_schedule() creates a schedule and adds it to database. Ask user to specify node id, action and time of execution.

Remove_schedule() – remove database entrance associated with schedule id

ChangeUsername() – allows to change a user name. Request: previous username, password and new username. If authentication parameters are correct => update username in database

changePassword() – allows to change a user password. Request: username, old password and a new password. If authentication parameters are correct => update password field in database

listGroups() – list all available groups from database

createAGroup() – creates a new group name in database

addNodeToAGroup() – associates a node with a group. Request: group name, node id(node name)

removeGroup() – remove group from database

removeNodeFromGroup() – remove association of node with a group from database

4.1.1.3 Process

All of the functions are encapsulated within a class and have no structured layout (no need, each function is completely independent component. Order is not important.)

4.2 Web Interface Subsystem

4.2.1 Description

The Web Interface consists of web pages served by the web server. What makes the interface distinct from the Web Server subsystem of the System Control Layer is that the Web Interface will consist of the server-side scripting, client-side scripting, and static HTML that makes up the graphical interface. This interface will not be involved in the operation of the system, but the presentation of it.

Specifically, the Web Interface is responsible for displaying all the needed information about the entire Z-Wave Smart Home Controller.

4.2.2 Interfaces

Each of the pages will interface between the user (via a web browser) and the web server. A combination of client-side scripting, HTML, and server-side scripting will allow the user interface with the SHC. Ultimately, all relevant Z-Wave web-server requests will be applied to the database for the System Control Layer to handle.

The Web Interface can be divided into 7 pages:

- Login Page
- Main Page
- Rules Page
- Schedules Page

- Map Page
- Groups Page
- Account Settings Page

These pages are described as follows:

4.2.2.1 Login Page

4.2.2.1.1 Description

The login page is responsible for authenticating the user. It is the landing page for the initial visit of the system via the web. It is also the page presented after logging out. The login page shall request a username and password, and provide a login button that submits any provided identification data to the web server for authentication. Once the user is authenticated, the Login Page requests the Main Page.

The login page is requested by all pages that require a currently logged in user. That is, each page will first request the login page to verify that a user is indeed logged in. If a user is logged in, the page loads normally, if not, the login page proper is displayed.

4.2.2.1.2 Process

General Page Request

```
If user is logged in then
     Display page requested, or main page if no page was requested
Else
     Display Login Form
```

Login Process

```
Display Login Form with username and password fields
Upon submission
     Check login information against Database records
     If user and password (or password hash) match
          Log user in, and then load the main page
     Else if they do not match
          Reload login form along with error message
End submission
```

4.2.2.2 Main Page

4.2.2.2.1 Description

The main page is the root of the ZSHC website. That is, the Main Page is the ZSHC website's index page. Upon requesting this page, the web interface will determine if the user is currently logged in or not. If the user is logged in, then the Main Page proper is loaded. All devices are enumerated and shown, as well as links to the other pages. This page is also responsible for handling voice control input. If the user is not logged in, the Main Page will request the login page be shown instead.

4.2.2.2.2 Data Structures

deviceList[]	The array of node objects in web presentation form (just enough information to display node data).
deviceControl	Object within deviceList[] that contains information about controlling the specific device it is a member of.
deviceGroups	Object holding an array of references to devices.

4.2.2.2.3 Process

```
Display if logged in:

     Device List with individual device controls tailored to device
type

     Links to Rules Page, Schedules Page, Map Page, and Account
Settings Page

Else

     Display Login Page
```

4.2.2.3 Rules Page

4.2.2.3.1 Description

The Rules Page is solely responsible for displaying all rules. It is also responsible for gather user input to add or remove rules and check on their statuses.

4.2.2.3.2 Process

```
Display if logged in:

     Rules List (enumerate all rules), each with remove rule option
```

```
     Links to Add Rule

Else

     Display Login Page
```

4.2.2.4 Schedules Page

The Schedules Page is solely responsible for displaying all schedules. It is also responsible for gathering user input to add or remove schedules and check on the schedules' statuses.

4.2.2.4.1 Process

```
Display if logged in:

     Schedule List (enumerate all schedules), each with remove
schedule option

     Links to Add Schedule

Else

     Display Login Page
```

4.2.2.4.2 Groups

4.2.2.4.3 Description

The group page will allow the user to group nodes into one or more groups. Groups are exclusive; no device can be added to more than one group.

4.2.2.5 Map Page

4.2.2.5.1 Description

The Map Page is solely responsible for displaying the 2D Map of the current ZSHC. It will allow the user to update the map by allowing input to add or remove devices and rooms.

4.2.2.5.2 Data Structures

mapData – XML SVG data that represents lines needed to draw rooms, as well as circles for individual devices.

4.2.2.5.3 Process

```
Display if logged in:

     Map loaded from database (HTML5 SVG)

     AJAX to handle map input
```

```
Else
     Display Login Page
```

4.2.2.6 Account Settings Page

4.2.2.6.1 Description

The Account Settings Page allows the user to perform administrative tasks related to the web interface, and the ZSHC. It will allow the user to change their username, password, and to log out of the web interface. It will also allow the user to backup the ZWSH data for use on any identical ZWSH product.

4.2.2.6.3 Data Structures

4.2.2.6.4 Process

```
Display if logged in:
     Links to Change Password, Change Username, Backup, Logout
Else
     Display Login Page
```

Change Password

```
Update database where username = username, and password =
new_password
```

Change Username

```
Update database where username = new_username, and password =
password
```

Backup

```
After confirmation, Export Database data to file, present file for
downloading.
```

Logout

```
End user session, return to main page.
```

4.3 Device Input Subsystem

Device Input is the subsystem that enables the user to handle the devices manually. It allows the user to turn on, off, or dim the light; turn the socket on or off; raise/lower the blinds, or open/close the slat. This subsystem consists of three modules: Light Switch, Socket, and BCU. These modules are responsible to control the lights, socket, and blinds, respectively.

4.3.1 Light Switch Module

Light Switch module operates on two push buttons. One button just lets you turn on or turn off the light, and the other one actually dim or brighten up the light. Once the hardware detects the status of buttons, it calls the Light Switch module of Light and Socket subsystem to control the light.

4.3.1.1 Interfaces

This module interfaces with the Light Switch module of Device Layer. It produces the integer value between 0 to 255 in order to command the Light and Switch module of the Device Layer to turn on or turn off the light. 0 represents the off stage of light, and 255 represent the on stage. And also based on the counter from the internal clock of the Light Switch module (hardware), it can produce the integer value (upto 255) to control the brightness of the light, which is also consumed by the Light Switch module of the Device Layer. The physical interface is two push buttons; one to just turn on or off the light, and the other push button to control the intensity of light.

4.3.1.2 Physical Data Structure/Data File Descriptions

- Boolean value (true and false) to turn the switch on and off respectively.
- Integer value ranging from 0 to 255 to control the brightness of the light.
- Electrical state of the switch (0 or 1) to enable the Boolean value.
- Internal clock to get the intensity of light.

4.3.1.3 Pseudocode Algorithm

//check the value of the buttons

//button1 is just to turn on and turn off the light, and button2 is to dim/brighten up the light

value1 = getValue(button1);

value2 = getValue(button2);

switch (value1);

dim (value2);

4.3.2 Socket Module

Socket module is responsible for controlling the socket manually. It depends on the push button as well. It functions in two stages only. It uses the same data type as in Light Switch module.

4.3.2.1 Interfaces

It interfaces with Socket module of Device Layer. The physical interface is the push button to turn on or off the socket. It sends the integer value after manual push button. As soon as the button is pressed, the internal clock is activated, which in turn produces integer value of 255. When the button is not pressed, the value remains 0.

4.3.2.2 Physical Data Structure/Data File Descriptions

It generates the integer value after manual push button. As soon as the button is pressed, the internal clock is activated, which in turn produces integer value of 255. When the button is not pressed, the value remains 0.

4.3.2.3 Pseudocode Algorithm

//get the value of the button.

value = getValue(button1);

switch (value);

4.3.3 BCU Module

BCU Module is responsible to control the blind based on manual input. The four buttons are created for controlling blinds. The modes of operation are raising the blind, lowering the blind, turn slat on, and turn slat off.

4.3.3.1 Interface

The physical interface is the four buttons, which links to the Event Handler module of the Blind Controller Unit subsystem. Buttons are arranged in the circuitry such that each button has separate bit sequence in the back end.

4.3.3.2 Physical Data Structure/Data File Descriptions

Each button produces the three bits string. Based on the bit sequence it will call appropriate function to control the blind. Following table shows what each bit sequence maps to the functionalities of blind.

011	Raise the blind

111	Lower the blind
110	Open the slat
100	Close the slat

4.3.3.3 Pseudocode Algorithm

```
//Verify the button, and generate respective bit sequence

if (button1.pressed())
{
     blindValue = 011;
}
if (button2.pressed())
{
     blindValue = 111;
}
if( button3.pressed())
{
     blindValue = 110;
}
if( button4.pressed())
{
     blindValue = 100;
}
eventHandler(blindValue);
```

5. Device Layer

The Device Layer contains the Blinds Control Unit (BCU), Z-Wave light switch, and Z-Wave electrical socket. These devices makeup a large portion of the hardware components found in the Z-Wave Smart Home (ZSH) System. This layer handles input from the user and status communication to the System Control Layer.

This layer consists of two subsystems, Blind Controller Unit subsystem, and Light and Socket subsystem. Blind Controller Unit subsystem is solely responsible for controlling blinds. Light and Socket subsystem takes care of functionalities of Z-wave enabled light and sockets. Both subsystems' operation is dependent upon the signal commands sent via Z-wave communication subsystem of System Control Layer. The signal commands can be either automated, or manual. Basically, this is the lowest level of the system, in terms of hardware.

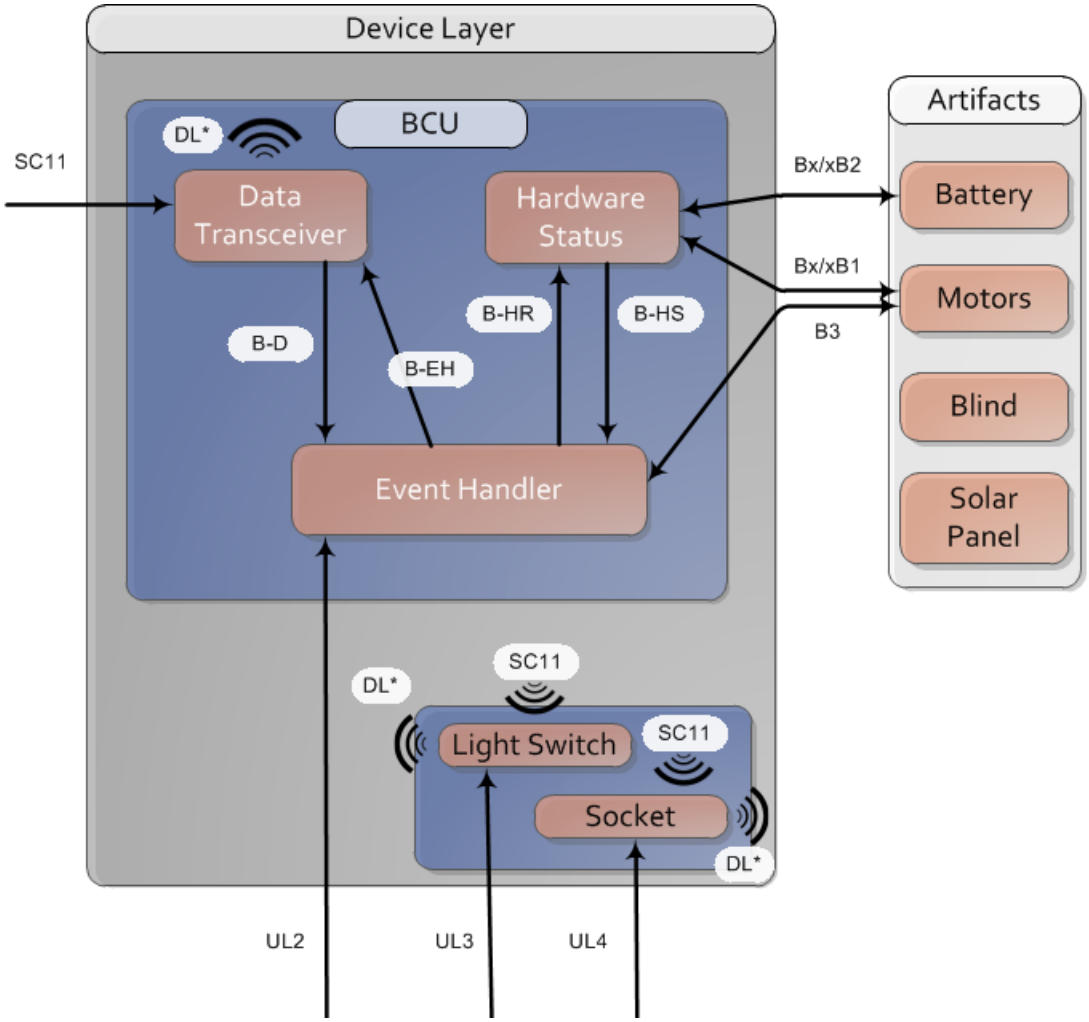


Figure 5-1: Device Layer

5.1 Blind Controller Unit Subsystem

This subsystem has five modules: Data Transceiver, Event Handler, and Hardware Status. This subsystem receives the RF packaged commands to control blind, light, and socket, and send back the status of these devices, to the system control layer.

5.1.1 Data Transceiver Module

Data Transceiver Module receives Z-wave commands embedded in RF signals. ZM2120C-E hardware - module is used to enable receiving Z-wave commands. It should be noted that ZM2120C-E is secondary controller, primary controller being USB dongle, and is set as a slave node. It is also set as Static Update Controller (SUC) to always act as a "listening node," and also act as repeater to associate with other nodes in the network. Controller Initiator is also activated in the slave node to connect with primary controller. After all the nodes have been established, and association has been made within nodes, Data Transceiver Module starts receiving the RF signals for further processing.

5.1.1.1 Interfaces

Data Transceiver Module interfaces with USB Dongle module and Event Recognition module. It consumes 64 bytes RF signals (beam), and transmits actual string data, including commands to be sent to Event Recognition module.

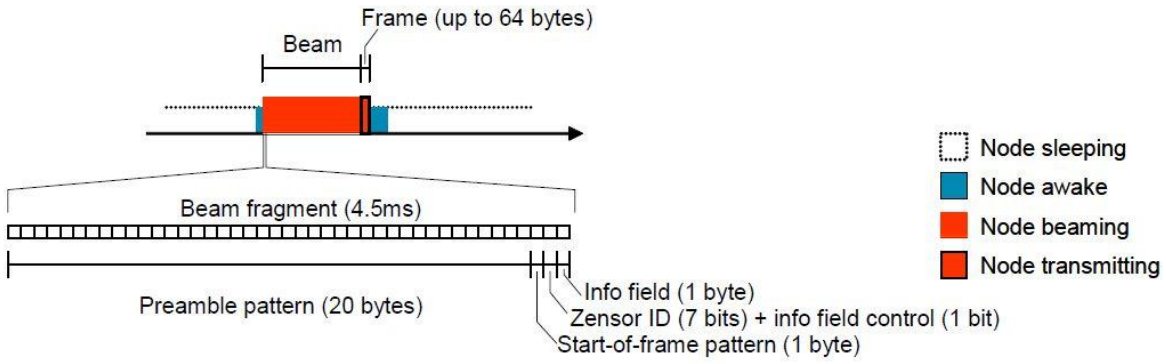


Figure 6-2: Structure of the beam

5.1.1.2 Physical Data Structure/Data File Descriptions

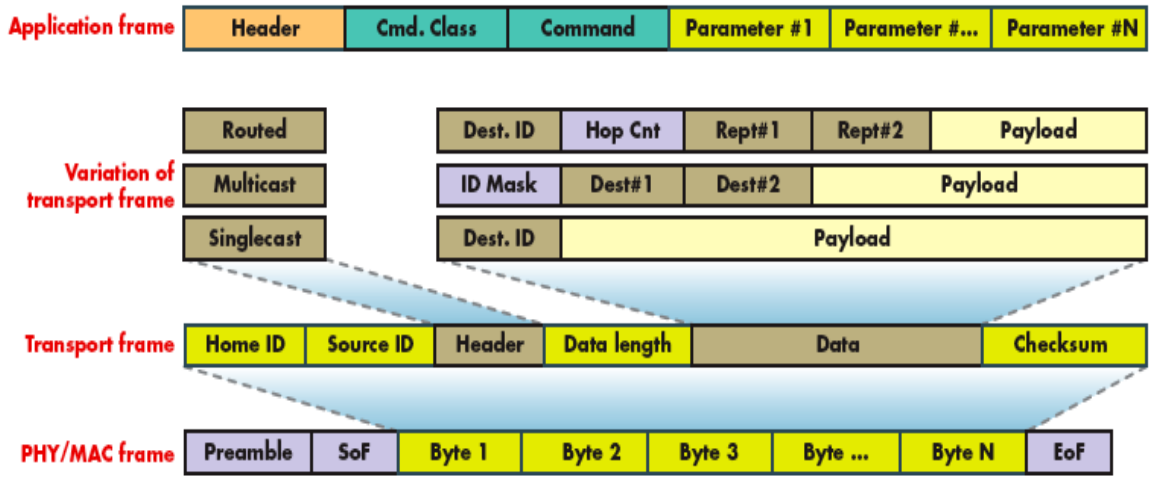


Figure: 6-3

Z-wave packets are divided into four frames. The above figure shows how packets are distributed. Application frame consists of actual data, in our case, the Blind control commands. The Transport frame consists of actual node ids and home ids to verify various modules in Z-wave network.

5.1.1.3 Pseudo-code Algorithm

All methods are based on Z-wave library APIs.

Check to see if it's receiving or transmitting data.

```
While receiving_once (true)
{
     //check for the information on node id, home id, and associated
     ids (light and socket //nodes) within the network.

     {
          If the primary controller node or any slave/associated
          nodes are not identified, send the information back.
          {
               transmit( ) ;
          }
          If nodes are identified, check for the stream of bits of
          data that has actual command to control the data.
          {
               string Command = parse_data(packets);
               //Send the command to the Event Recognition module

          }
          If receive( ) is the node id in the network, relay the
     information to USB dongle
          {
```

```
                transmit( ) ;
           }
     }
}
If transmiting_data (packets)
{
     transmit(packets);
}
```

5.1.2 Event Handler

Event Handler module is responsible for recognizing actual command, and executing those commands to control the blind.

5.1.2.1 Interface

After Data Transceiver filters the signals, it sends the string data, which is consumed by the Event handler module. This module then recognizes the actual command based on the stream of bits in the string so that it can produce specific command like raise/lower the blinds, or rotate the slat, or just get the battery level. It also interfaces with Hardware Status module by a function call requesting the status of hardwares. .

5.1.2.3 Pseudo-code Algorithm

check the stream of bits and produce specific commands

```
Event eventHandler()
{
switch (command);
{
Case “000”:
//do nothing
Break;

Case “001”:
//request battery level
Poll_battery();
break;

Case “011”:
//raise blinds
currentStatus= motor1Status();
if (currentStatus==false)
{
     Raise(motor_count);
     currentStatus=true;
}
Else do nothing;
Break;
```

```
Case “111”:
//lower blinds
currentStatus= motor1Status();
if (currentStatus)
{
     lower(motor_count);
     currentStatus=false;
}
Else do nothing;
Break;

Case “110”:
// open slats
currentStatus2= motor2Status();
if (currentStatus2==false)
{
     Rotate(motor_count);
     currentStatus2=true;
}
Else do nothing;
Break;

Case: 100
//close blinds
currentStatus2= motor2Status();
if (currentStatus2)
{
     Rotate_anti(motor_count);
     currentStatus2=false;
}
Else do nothing;
Break;

Default:
//return invalid message
Break;
}
}
```

5.1.2.2 Physical Data Structure/Data File Descriptions

Event Handler receives the string of data as a command either from data transceiver module or BCU module of the device input subsystem of user interface layer. The string is 3 bits sequence, and each sequence signifies the command. It also requests the hardware status of blinds and the battery level and sends commands to the motors for the proper execution of the commands received from the data transceiver. The status of the blind is represented using Boolean value.

5.1.3. Hardware status

5.1.3.1 Interface

Hardware status is responsible for getting the current status of the hardware devices namely battery and motors. And send this status to the event handler for the proper execution of the commands.

5.1.2.3 Pseudo-code Algorithm

```
check the stream of bits and produce specific commands
//check  the status of battery
Poll_battery()
{
//Convert 8 bit data received to voltage that ranges from 0 to 5 volts
//if voltage level<1V turn the LED signal on. It means the battery is low
//else do nothing
}
;

//check the status of the motor1 which is responsible for setting the height of the blinds
Public 40oolean motor1Status()
{
if (getCount(motor1))
return true;
else
return false;
}


//check the status of the motor1 which is responsible for opening and closing of the slats
Public 40oolean motor2Status()
{
if (getCount(motor2))
return true;
else
return false;
}
```

5.1.2.2 Physical Data Structure/Data File Descriptions

The hardware status module uses the motor API to retrieve the status of the motor. The value of the count ranges from 0 to 65536 as it is using integer type. Practically, the count depends on the PWM signal so it would be calibrated accordingly. Similarly, it gets the 8 bits integer data after ADC conversion to represent the voltage.

5.2 Electrical Socket & Light Switch Subsystem

5.2.1 Socket Module

The Socket Module is comprised of GE 45605 Z-Wave Wireless Lighting Control Duplex Receptacle and is responsible for controlling electricity to appliances. This module only controls the electricity flow of one socket via Z-Wave commands while the other socket has power at all times. This module relies on Z-Wave protocols built into the electrical socket. This module also requires a standard input voltage of 120 volts as commonly found in U.S. households.

5.2.1.1 Interfaces

This module interfaces with the socket module in the Device Interface Subsystem within the User Layer. The data received is in the form of an integer command based on the press of a button on the device.

This module interfaces with the USB dongle module in the Z-Wave Adapter Subsystem within the System Control Layer. The data received from the USB dongle module is in the form of a standard Z-Wave protocol packet. This is a series of 64 bytes transmitted via radio frequency which contains information pertaining to network ID, device ID, and standard commands.

This module interfaces with the Data Transceiver module in the BCU Subsystem within the Device Layer. The data sent/received from the BCU module is in the form of a standard Z-Wave protocol packet. This is a series of 64 bytes transmitted via radio frequency which contains information pertaining to network ID, device ID, and standard commands.

5.2.1.2 Physical data structure/data file descriptions

Externally, the socket module is dependent on the Z-Wave protocol packet. This packet is 64 bytes long of which the module is mostly concerned with six bytes in the application frame of the packet. This frame contains the header, command class, command, and three parameter values all of which are one byte long.

Internally, the socket module uses the command class and command byte codes to execute a predefined function. These functions are based on proprietary Z-Wave function calls for off/on operation.

5.2.1.3 Process

Constantly listening for packets

Packet received

Inspect packet for node ID

Determine if module node ID and packet node ID match

If node IDs match: perform command operation

Else: broadcast packet to nearest neighbors

Example command:

```
sub switch {
    my( $unit, $onoff ) = @_;
    dim( $unit, $onoff ? 255 : 0 );
}
```

5.2.2 Light Switch Module

The Light Switch Module consists of the Leviton VRI06-1LZ Vizia RF 600W Incandescent Scene Capable Dimmer and is responsible for controlling the brightness of a light bulb. This module only controls the electricity flow to a light bulb via Z-Wave commands. This module relies on Z-Wave protocols built into the electrical socket. This module also requires a standard input voltage of 120 volts as commonly found in U.S. households.

5.2.2.1 Interfaces

This module interfaces with the light switch module in the Device Interface Subsystem within the User Layer. The data received is in the form of an integer command based on the press of a button on the device. There are three buttons one of which controls the on/off function and two of which control the dimming of the light.

This module interfaces with the USB dongle module in the Z-Wave Adapter Subsystem within the System Control Layer. The data received from the USB dongle module is in the form of a standard Z-

Wave protocol packet. This is a series of 64 bytes transmitted via radio frequency which contains information pertaining to network ID, device ID, and standard commands.

This module interfaces with the Data Transceiver module in the BCU Subsystem within the Device Layer. The data sent/received from the BCU module is in the form of a standard Z-Wave protocol packet. This is a series of 64 bytes transmitted via radio frequency which contains information pertaining to network ID, device ID, and standard commands.

5.2.2.2 Physical data structure/data file descriptions

Externally, the light switch module is dependent on the Z-Wave protocol packet. This packet is 64 bytes long of which the module is mostly concerned with six bytes in the application frame of the packet. This frame contains the header, command class, command, and three parameter values all of which are one byte long.

Internally, the light switch module uses the command class and command byte codes to execute a predefined function. These functions are based on proprietary Z-Wave function calls for off/on operation and dimming. The dimming function will take on a parameter value to indicate the level of brightness the bulb should produce based on a value between 0 and 255.

5.2.2.3 Process

Constantly listening for packets

Packet received

Inspect packet for node ID

Determine if module node ID and packet node ID match

If node IDs match: perform command operation

Else: broadcast packet to nearest neighbors

Example commands:

```
sub switch {
    my( $unit, $onoff ) = @_;
    dim( $unit, $onoff ? 255 : 0 );
}
sub dim {
    my( $unit, $level ) = @_;
    return( packpack( mkreqpacket( 0, 0x13, $unit, 3, 0x20, 1,
$level, 5 ) ) );
}
```

6. Quality Assurance

6.1 Test Plans and Procedures

6.1.1 General

The ZSH system shall be put through a series of test as Team betahomes desires to deliver the highest possible quality product within the given constraints. These tests shall confirm if the architecture meets the desired design requirement specification. All the layers and their interactions will be taken into consideration for this purpose. Testing will include unit testing, system testing, and integration testing. Inputs and outputs will be validated by passing in valid inputs to get the desired outputs and passing in invalid inputs to make sure the system terminates gracefully.

Each layer will first be tested independently by performing the unit testing. Once it passes the unit testing it will be tested with other layers and subsystems upon integration which will be done through integration testing. Integration testing will ensure that each layer interacts with each other as expected. Finally the system verification testing will be conducted to ensure all the customer requirements are met and the system is robust.

6.1.1.1 Module/Unit Testing

Z-Wave Server:

- Verify if the Z-Wave network can be scanned correctly.
    - The verification above handled by running server_1.py file which creates xml file with nodes information. If file been created successfully, therefore Z-Wave network is valid/exist. If not, then either network is not accessible or the server_1.py file which communicates with open Z-Wave libraries wasn't implemented correctly.
- Verify if the status of nodes can be updated correctly.
    - Execute an action/command using perlscript
- Verify if the database is updated correctly.
    - Visual contact with a database through phpmyadmin.
- Verify if the execution command is passed correctly to the perl script.
    - Status of the nodes updated through PerlScript. The verification of this case is pretty straight forward. There are two options we have. Option 1: check status of the node by running Open-ZWave test server application. All it does is intercepts all outgoing signals from all nodes. From here we can take a sample generated output and filter it for a specific node we are looking for. Since we do know the structure of the Z-Wave protocol (this information been accumulated during long period of research and datasheets review), we can get all the information regarding node content. Option 2: Using

PerlScript we can just listen for a specific output from Z-Wave Network. Specifically – "0, 0x4a, 0x01" string, which means that recent action been successfully processed by a node. Option 3: (easiest option) Visual contact with a node.

Nodes Inspector

- Verify if the home nodes network are scanned correctly every 3 hours to make sure they all function correctly.
    - Check the creation time of the xml file generated by inspector.py
- Verify if the current node home network xml map is removed in case of death of any node.
    - Visual contact with an xml file. Check the date and time file been created.
- Verify if a new xml map is recreated with updated nodes information in case of death of any of the nodes.
    - Visual contact with an xml file. Check the date and time file been created.

PerlScript

- Verify if devices can be added to the Z-Wave network using the "zwave add" command.
    - Can be verified during creation of xml home network map which contains information of all nodes and their data.

Open-ZWave Libraries

- Verify by making sure if the Z-Wave controller can be controlled as desired by using the API's provided by the Open-ZWave libraries.
    - Check with a Open-ZWave library list of supported USB devices(adapters) or (if not listed) test yourself using "zwave add" command provided by PerlScript.

USB Dongle Module

- Verify if the USB dongle is detected.
    - By running sample Open-ZWave server application we can see if driver for the USB device been installed. If yes, therefore adapter is detected.

Database module

- Verify the results retrieved from the database are correct.
    - Send test queries.

Data transceiver module

- Verify if it is receiving valid z-wave packets
    - By calling the receive_once function and getting either invalid message, as it could not connect to the network, or parsing the command data out of package.

Event handler module

- Verify if the blind is opening and closing, and the slat is rotating.
    - By sending the command signals to control the blind and slat. If it responds to the commands sent by the Data Transceiver module, then that means the event handler module is working properly.

Hardware status module

- Verify if the module returns the correct status of the blind, and battery level.
    - By sending the getStatus commands, and comparing the result visually.

6.1.1.2 Integration Testing

We will now discuss how each layer and its modules will be tested in order to verify our architectural design and validate our system requirements.

User Layer

The user layer provides an interface for interaction between the user and the system through web interface and manual input from the device. The User Layer will be tested by verifying the interactions between its various GUI interfaces. The outputs generated in response to various inputs will be matched with the expected results. The web server will be tested using the web interface. For instance when the user enters the user id and password, the web server should deliver the main page if the login is successful or display the login error page if the login fails. Black box testing will be conducted to make sure all the results are as expected.

Device Layer

The device layer consists of the main hardware components namely Blind Controller Unit, the Light and the Socket. The testing of the device layer will be white box testing. The device layer will be tested using the user layer that includes both web interface and device input. Testing will be conducted to make sure all the instructions are performed as requested by the user. For instance, when the user sends in signal to open the blinds, the blinds should rotate accordingly.

System Control Layer

The system control layer is the most critical layer of the system. It consists of z-wave communications subsystem, data processor, operating system and database. The database will be tested by sending test

queries using the web server. The data processor sub system will be tested by sending signals from the web server and making sure appropriate response is generated. Z-Wave communication subsystem will be tested using the device layer by making sure the devices respond as per the commands sends to the z-wave communication subsystem.

6.1.1.3 System Verification Test

The System Verification Test will verify the ZSH System has implemented all the high priority requirements specified by the customer in the SRS.

7. Requirements Traceability Matrix

7.1 Overview

The purpose of the requirement mapping is to give an overview of the requirements specified in System Requirement Specification that are intended to be satisfied based on the Architecture Design Specification's subsystems. It also traces the relationship between the requirements in the System Requirements Specification and the actual functions and modules that satisfy these requirements.

7.2 Mapping

Number	Requirement	System Control Layer	User Layer	Device Layer
3.1	Web Interface		✓	✓
3.2	Automated Device Control	✓	✓	✓
3.3	Scheduling		✓	✓
3.4	Rules		✓	✓
3.5	Manual Device Control		✓	✓
3.6	Modularity		✓	
3.7	Voice Control		✓	✓
3.8	2D Map		✓	✓
3.9	Z-Wave Device Communication	✓		✓
3.10	Z-Wave Controller Communication	✓		✓
3.11	Blinds Control Unit Slat Elevation	✓	✓	✓
3.12	Blinds Control Unit Slat Rotation	✓	✓	✓
3.17	The application has a meaningful GUI		✓	

3.18	Device Status Feedback	✓	✓	✓
3.19	Blind Control Unit Manual Control		✓	✓

Figure 7-1: Architecture Requirements Mapping

7.3 Modules Requirements Mapping

Modules	Requirements														
	Web Interface	Automated Device Control	Scheduling	Rules	Manual Device Control	Modularity	Voice Control	2D Map	Z-Wave Device Communication	Z-Wave Controller Communication	Blinds Control Unit Slat Elevation	Blinds Control Unit Slat Rotation	The application has a meaningful GUI	Device Status Feedback	Blind Control Unit Manual Control
Functions Class Library	X	X	X	X			X	X					X	X	
LogIn / Authentication	X														
User Interface	X	X	X	X			X	X					X	X	
Light Switch (Device i/p)					X	X									
Socket					X	X									
BCU					X	X					X	X			X
USB Dongle									X	X				X	
Z-Wave Server		X												X	

Nodes Inspector		X											X	
PerlScript (Executor)									X	X	X	X	X	
Open-ZWave Libraries									X	X			X	
Database	X	X	X	X			X	X					X	
Data Transceiver		X							X	X	X	X	X	
Hardware Status	X				X				X	X	X	X		
Event Handler	X	X			X				X	X	X	X		X
Light Switch (device layer)	X	X	X	X	X				X	X			X	
Socket	X	X	X	X	X				X	X			X	

Table 7-2: Module Requirements Mapping

7.4 Producer/Consumer Relationships

The producer-consumer relationships define the communication paths between modules within the system. The following table identifies the relationships which are detailed further by referring to dataflow section defined earlier.

We can see that the Producer Consumer Relationship figure depicts the data elements' flow within modules. Here, rows denote the producer modules from where data generates, and columns denote the consumer modules which receives the data.

Producer – Consumer Relationship		CONSUMER SUBSYSTEM								
		Web Interface (User Layer)	Device Input (User Layer)	Blind Controller Unit (Device Layer)	Light and Socket (Device Layer)	Z-Wave Communications (System Control Layer)	Data Processor (System Control Layer)	Operating System (System Control Layer)	Web Server (System Control Layer)	Database (System Control Layer)
PRODUCER SUBSYSTEM	Web Interface (User Layer)								UL1	
	Device Input (User Layer)			UL3	UL2					
	Blind Controller Unit (Device Layer)				DL4	DL1				
	Light and Socket (Device Layer)			DL3		DL2				
	Z-Wave Communications (System Control Layer)			SC11	SC13		SC1			
	Data Processor (System Control Layer)					SC2		SC10	SC5	SC3
	Operating System (System Control Layer)						SC9		SC8	
	Web Server (System Control Layer)	SC12					SC6	SC7		
	Database (System Control						SC4			

M	Layer)									

Table 7-3: Producer Consumer Relationship

8. Acceptance Plan

8.1 Overview

This section discusses the acceptance criteria that must be met by the ZSH system to be considered minimally complete. These criterions are critical and must be fulfilled in order for the end product to be accepted by the customers and the stakeholders.

8.2 Packaging and Installation

The ZSH System will contain a blinds control unit, light switch module, electrical socket module, CD with backup software, mini Linux server, Z wave USB controller and a user manual. The user manual shall provide all the instructions required for the installation and maintenance of the ZSH system.

8.3 Acceptance Testing

ZSH System acceptance testing shall be conducted to ensure that product meets the entire acceptance criterion. System testing shall be conducted to ensure that the system's performance meets the customer expectation. The details of this testing will be provided in the System Test Plan document.

8.4 Acceptance Criteria

Acceptance criteria are the requirements that must be completed for the project to be accepted as complete. These requirements include the top priorities of the project which are derived from the critical requirements and would affect the functionality of the product if not taken into consideration.

ZSH System must meet the following requirements agreed upon by all the stakeholders involved in the project:

- The product is web based
- The controller is Z-wave enabled
- The system is modular
- The product has no sharp edges
- The system is properly insulated
- The system has no hanging wires
- The modules are z-wave enabled
- The application has a meaningful GUI

9. Appendices

- Z-Wave Command Classes
- Software Design Specification: Z-Wave Protocol Overview
- Z-Wave Node Type Overview and Network Installation Guide